sPHENIX

Pileup Tracking Simulations in Au+Au

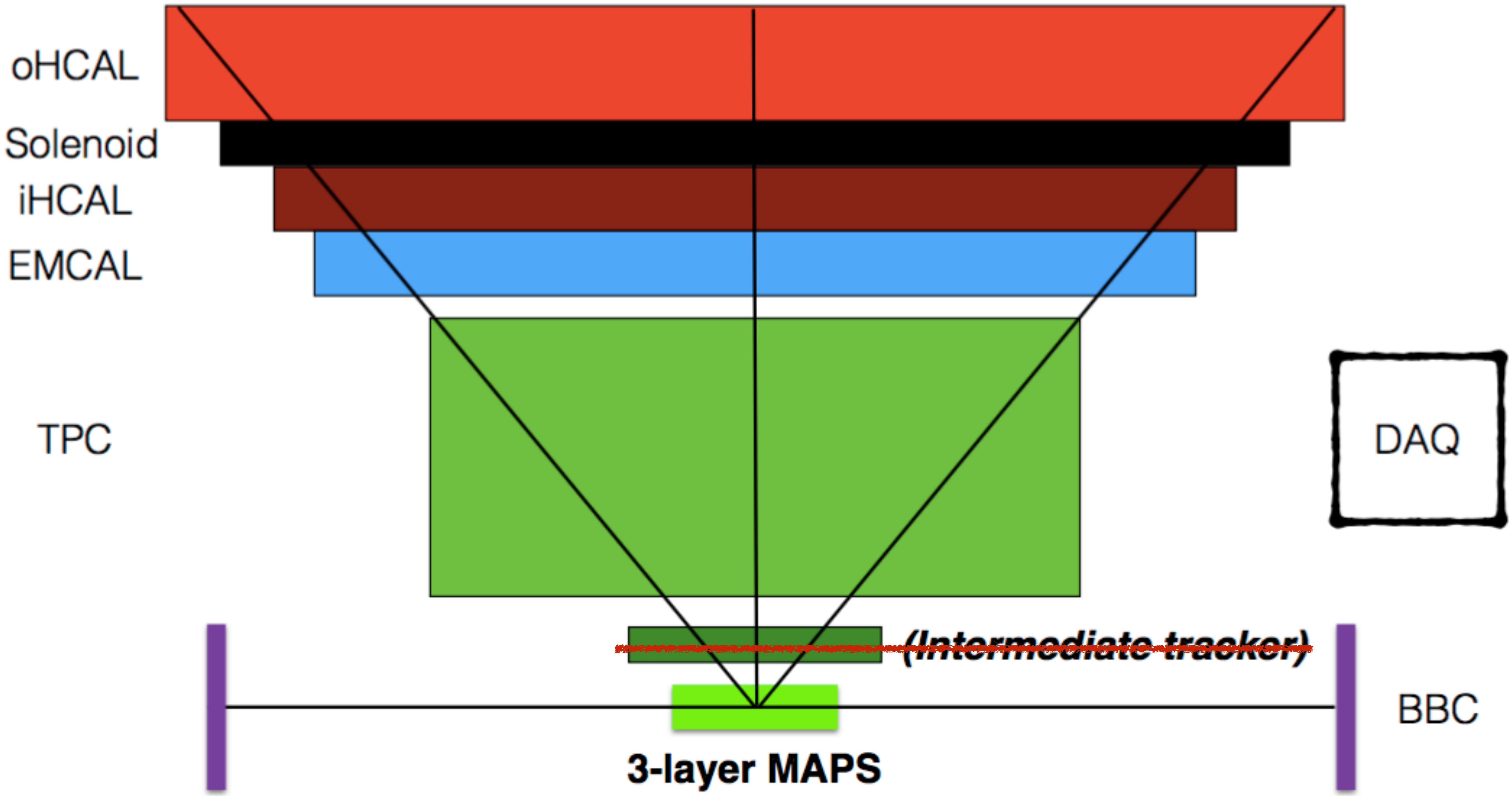
Michael P. McCumber

Simulations Meeting

October 4th 2016

New sPHENIX Baseline

for today I show only 3 Layers of MAPS + TPC



Reference configuration

Basic Numbers

Number of crossings during the roughly calculated integration windows:

MAPS +/- 2 us => 37 crossings can contribute hits

TPC +/- 18 us => 340 crossing can contribute hits

Peak Luminosity estimates

p+p => 2000 kHz => 0.212 chance of an interaction per crossing

Au+Au => 100 kHz => 0.011 chance of an interaction per crossing

MAPS:

p+p 8 events of pileup **<= peak occupancy for vertexing**

Au+Au 0.4 events of pileup

TPC:

p+p: 72 events of pileup

Au+Au: 3.6 events of pileup **<= peak occupancy for tracking**

Questions:

Au+Au: How many inner space points (MAPS) are needed to confirm a TPC stub?

p+p: How well can we multi-vertex?

Modified Pileup Tracking Performance Test

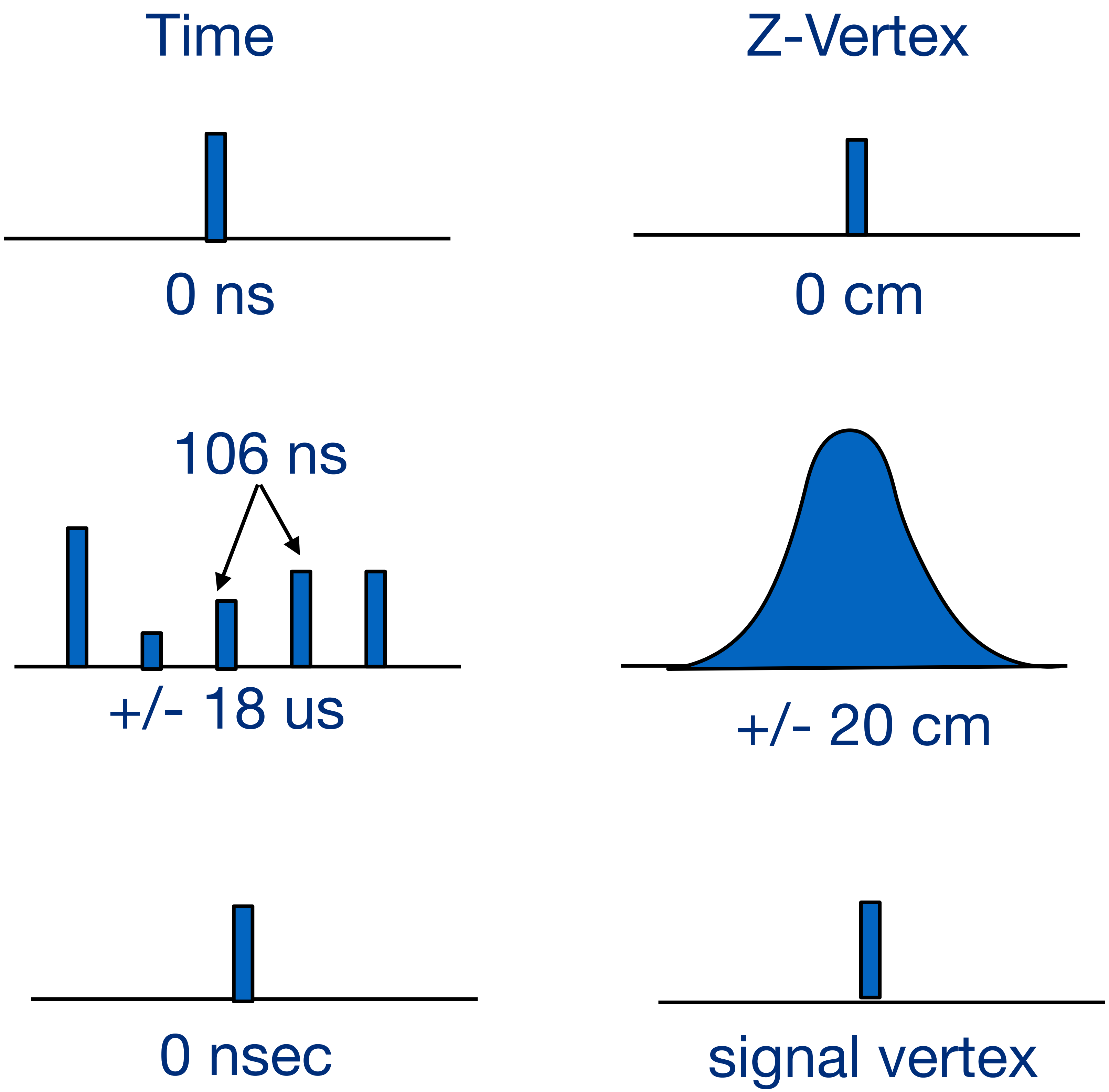
Single Central 0-4 fm Au+Au (HepMC)

+

Multiple MinBias 0-14 fm Au+Au (Pileup Input Manager)

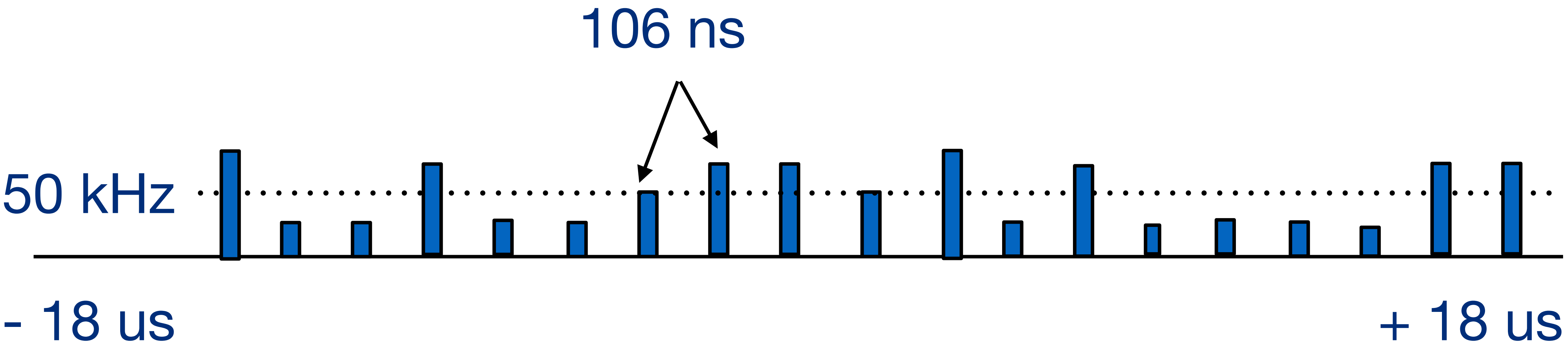
+

20 embedded pions (Simple Event Generator)

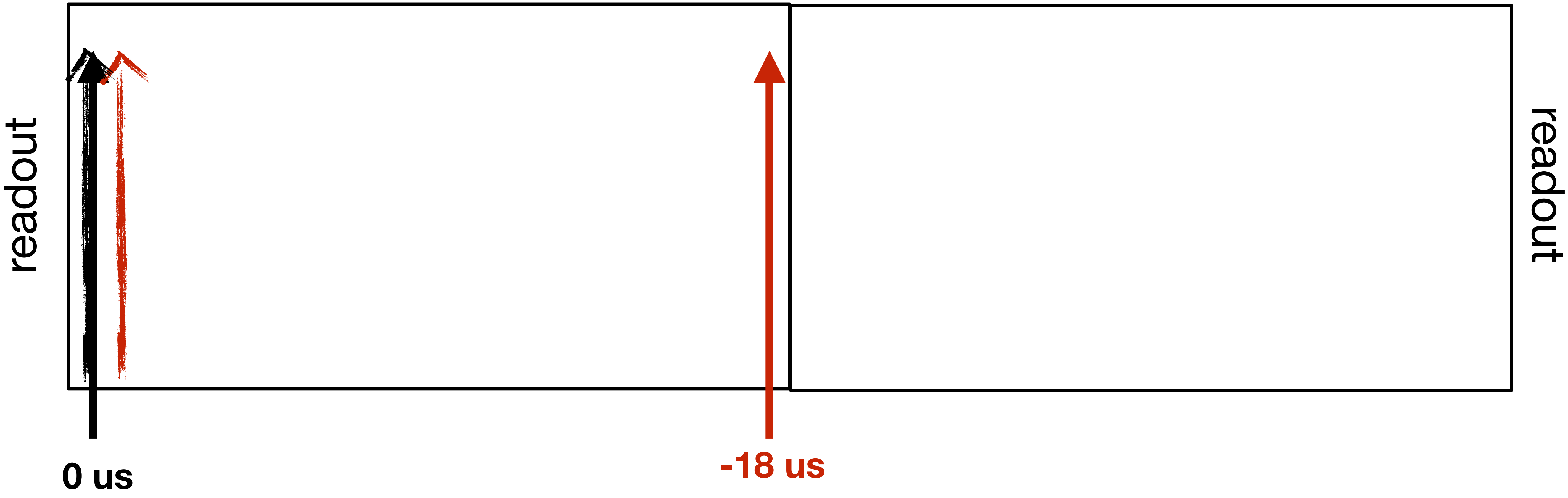


Brute force the vertex finding to the correct value.

Pileup Time Structure

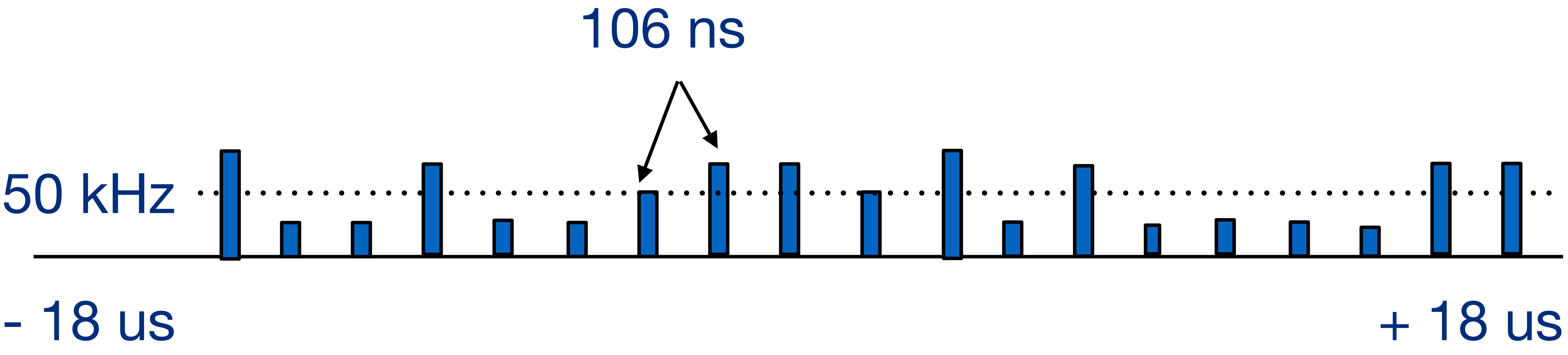


readout window will be trigger at 0 sec + 18 us

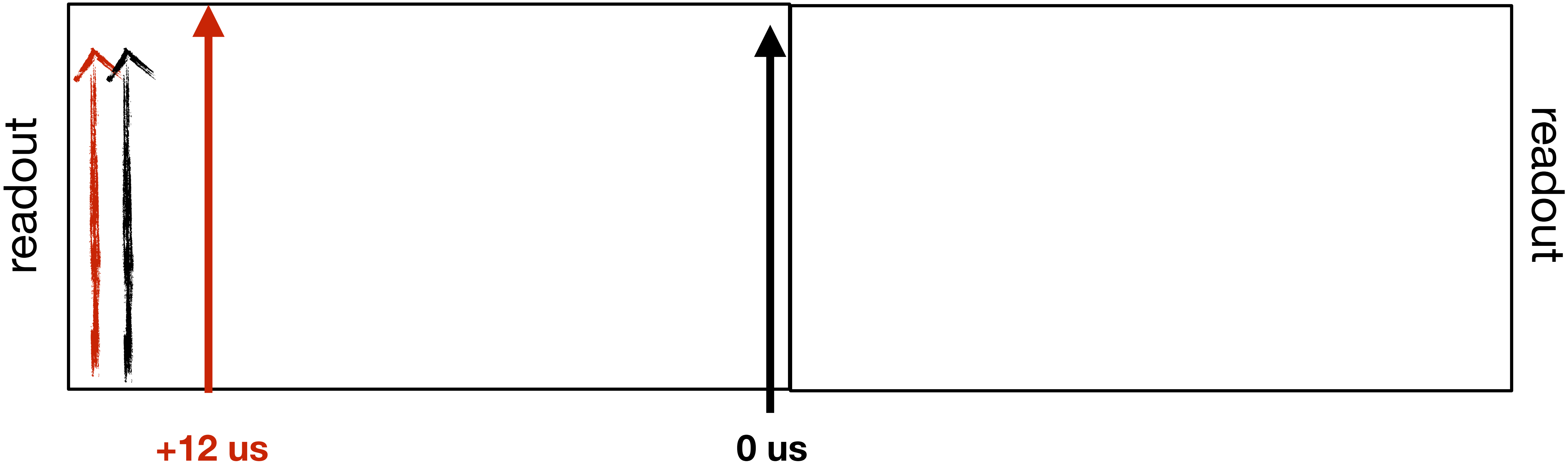


past time pileup from things already drifting to the readout

Pileup Time Structure

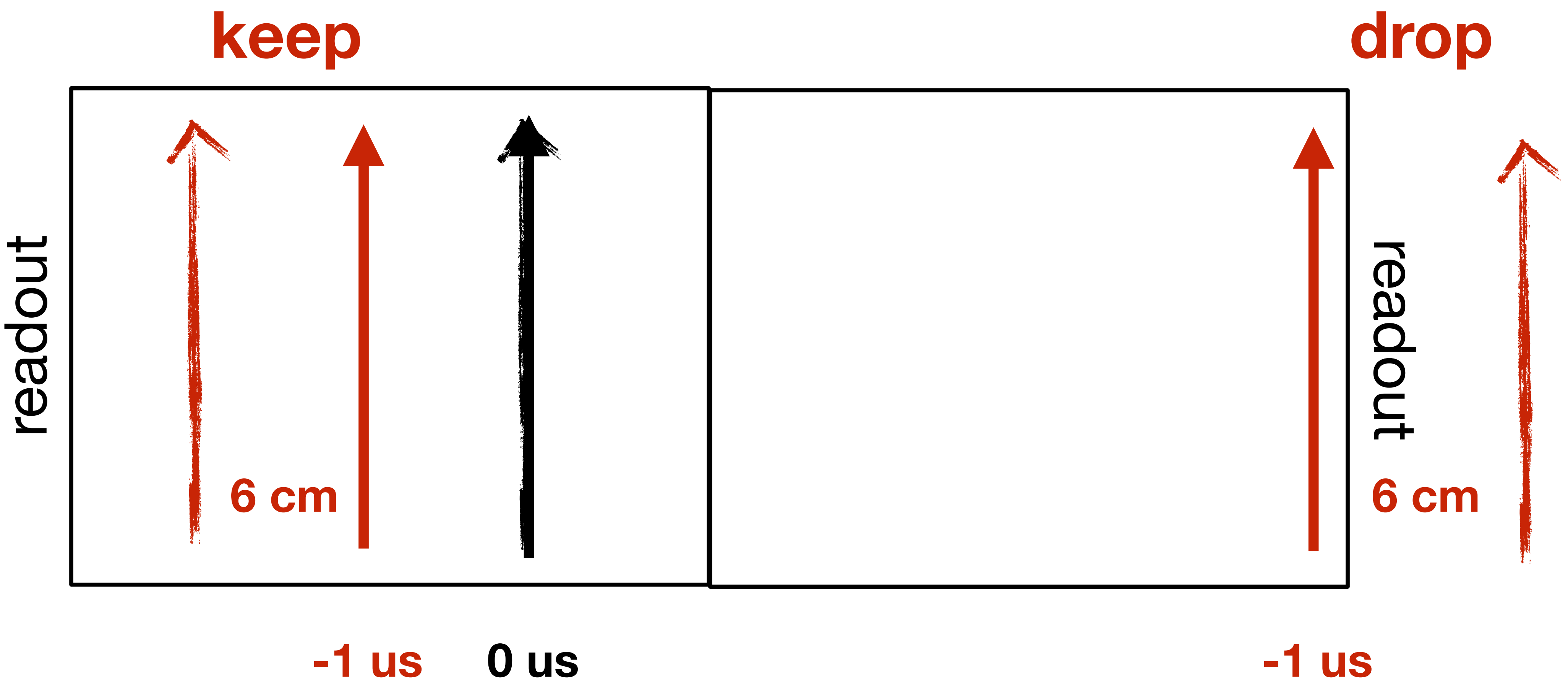


readout window will trigger at 0 us and continue for 18 us



future time pileup from signals created during the drift time

TPC Drift



It is possible for some signals in the +/- 18 us window of pileup generation to be "drifted" outside the 1/2 TPC volume

These are dropped as we would know by the time arrival that they are unassociated with the current trigger.

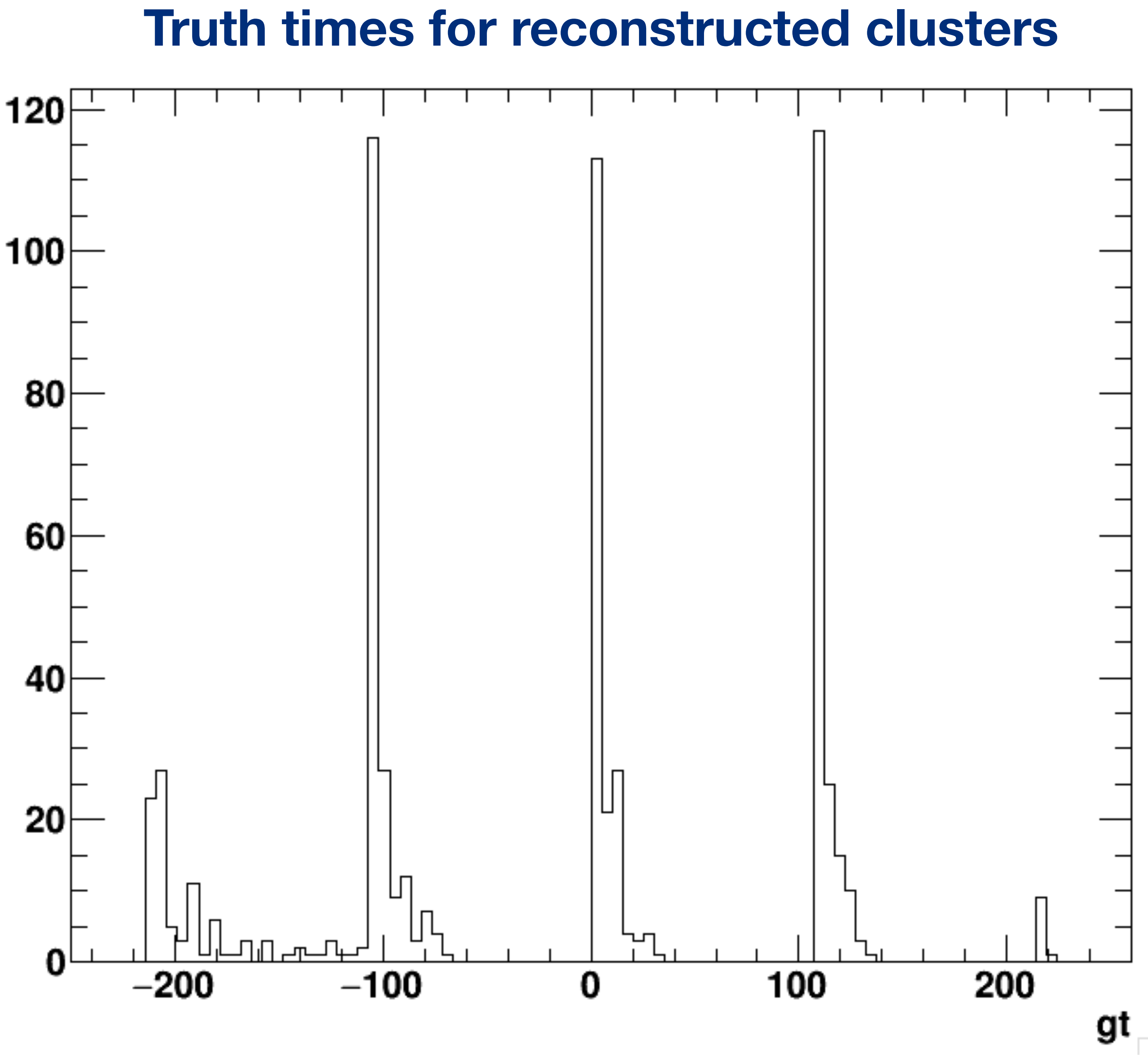
This prevents over-estimating the TPC occupancy.

Bug Fix: Before

My original attempt had an interference between the diffusion and the drift.

There was a call back to the original z-location late in the diffusion calculation.

The net effect diffused off-time clusters into oblivion leaving only a few crossings contributing to TPC occupancy.



Bug Fix: After

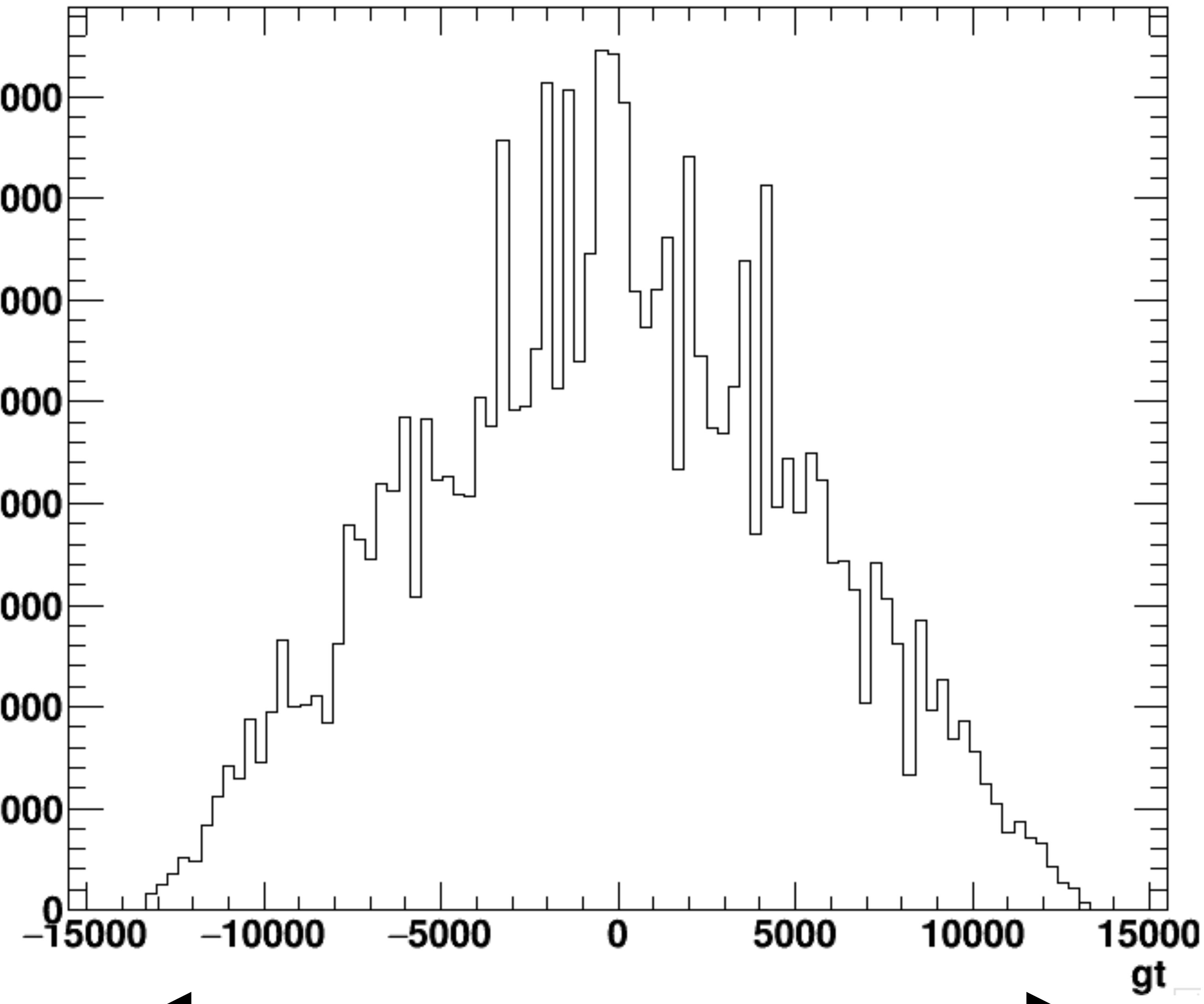
My original attempt had an interference between the diffusion and the drift.

There was a call back to the original z-location late in the diffusion calculation.

The net effect diffused off-time clusters into oblivion leaving only a few crossings contributing to TPC occupancy.

After:

Truth times for reconstructed clusters

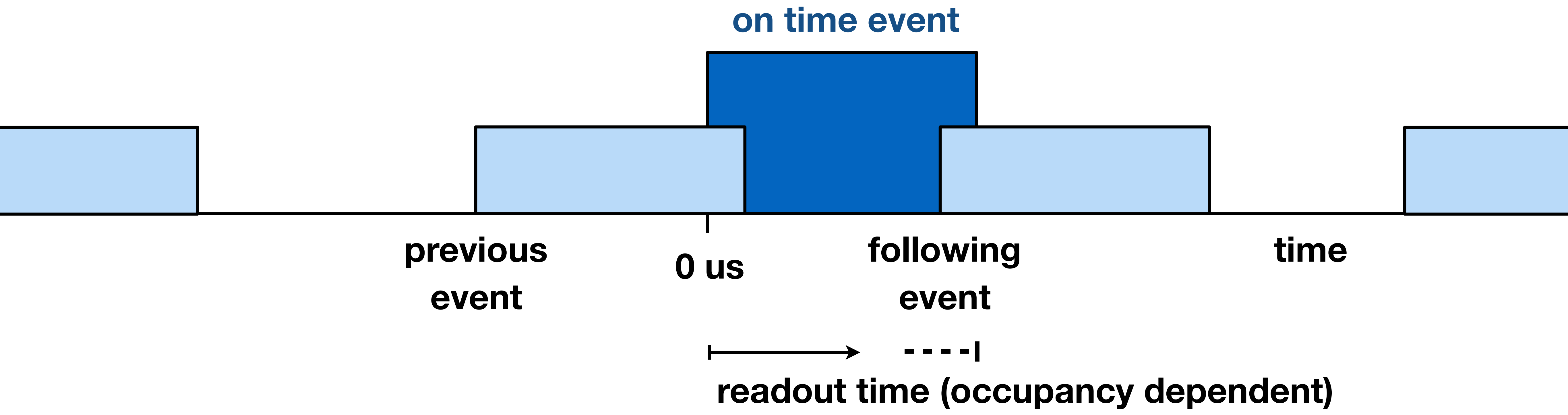


already drifting before trigger

deposited during integration time

MAPS Pileup

Struck pixels rise quickly, but stay above threshold for 2 us

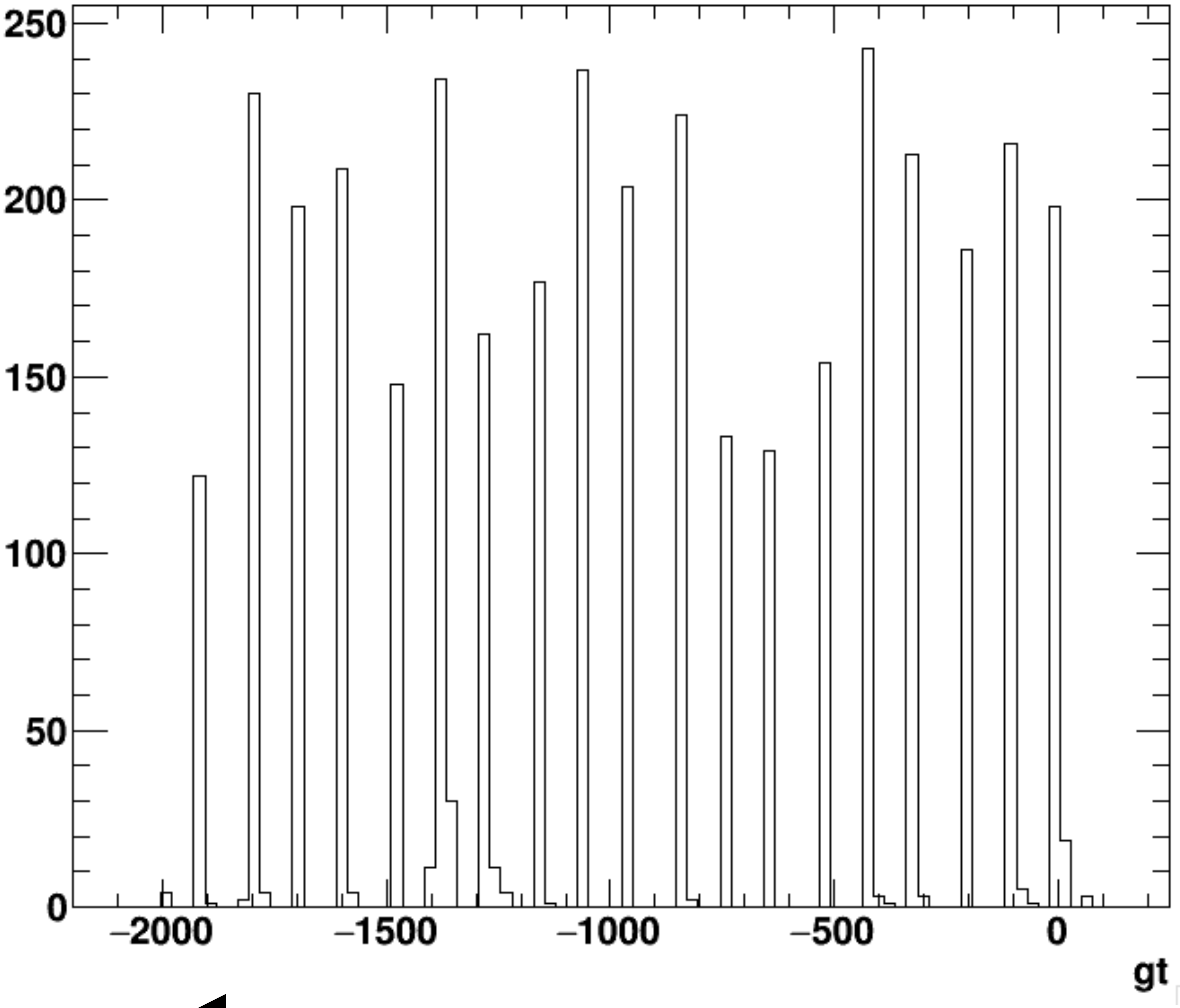


I model the readout time as instantaneous. Only events during the past 2 microsecond can contribute. In reality, the readout duration will bring in some future collisions and drop and equal number of past collisions (smear this integration across time).

This way I get the right average hit occupancy but underestimate the number of collision vertexes.

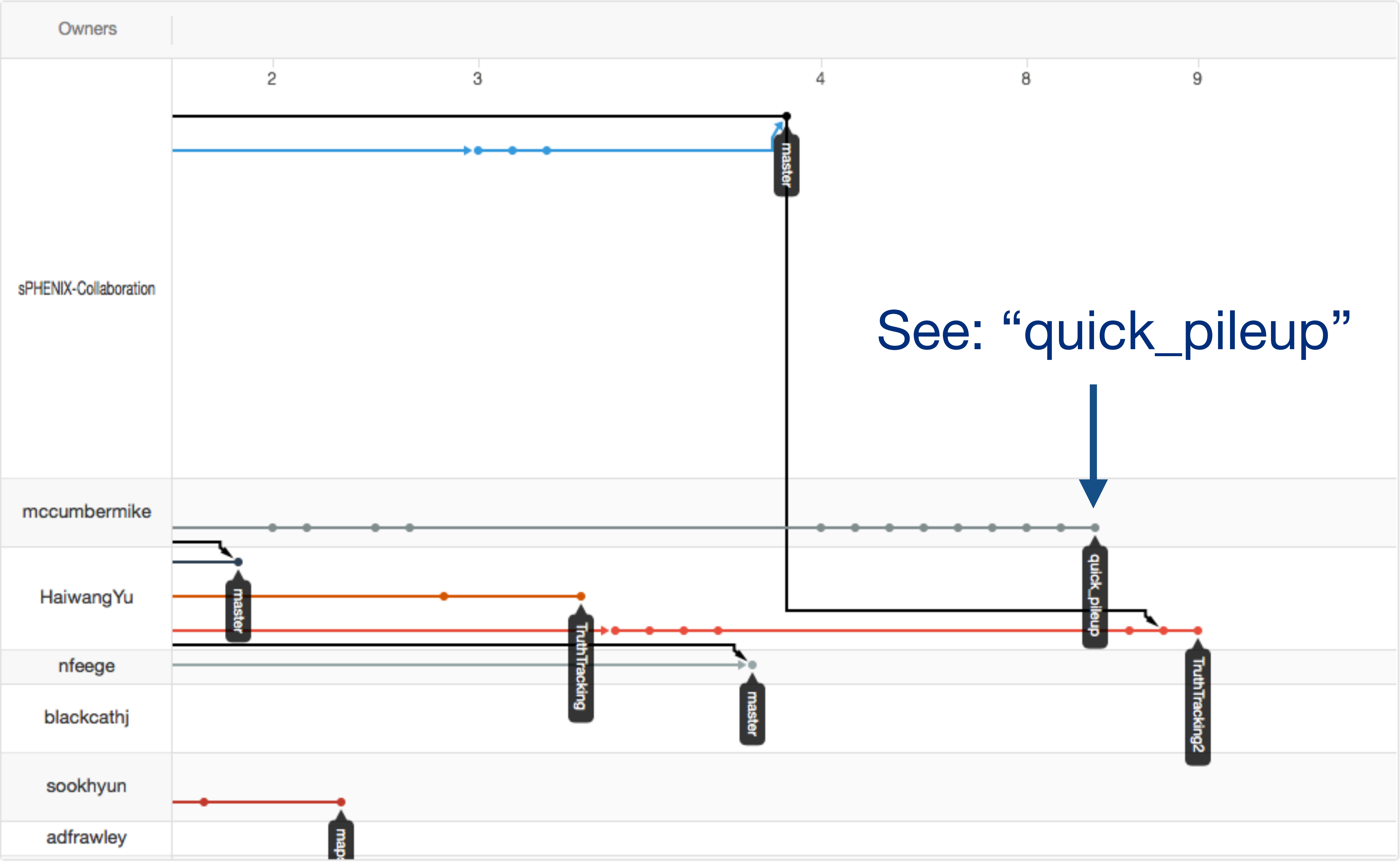
MAPS Timing

Truth times for reconstructed clusters



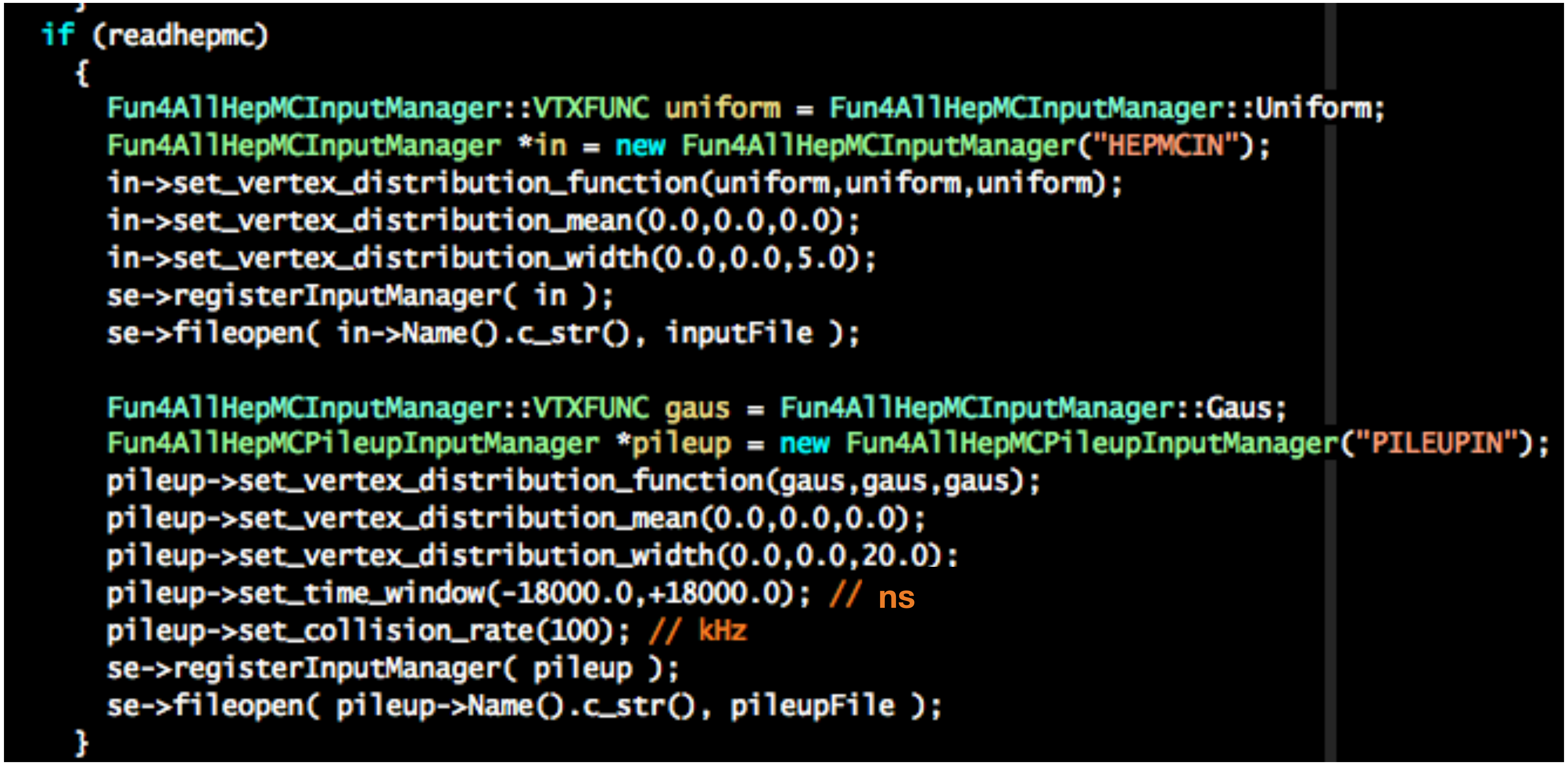
integration over past 2 usec

Pileup Branch



See: “quick_pileup”

quick_pileup Usage



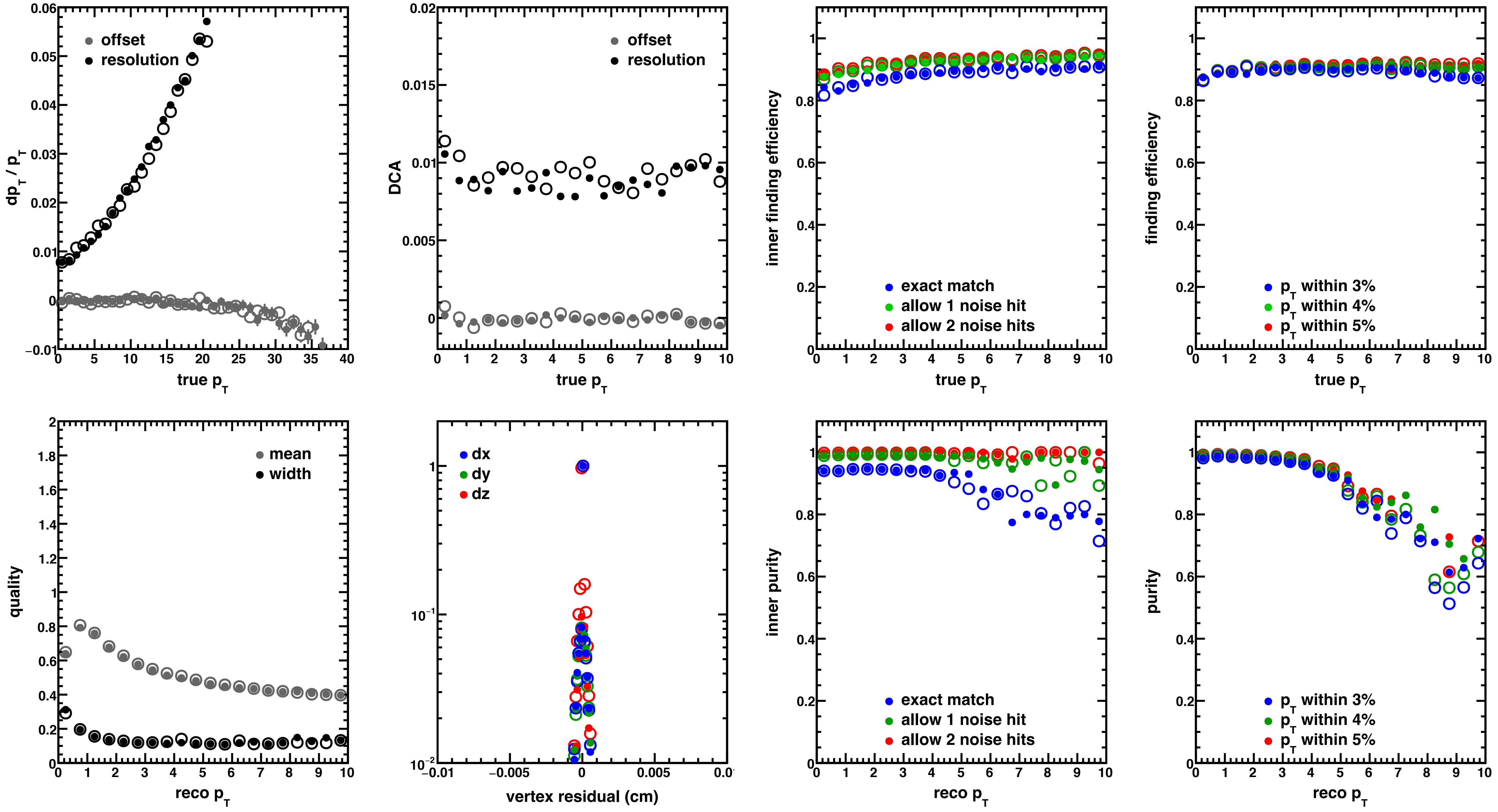
```
if (readhepmc)
  {
    Fun4AllHepMCInputManager::VTXFUNC uniform = Fun4AllHepMCInputManager::Uniform;
    Fun4AllHepMCInputManager *in = new Fun4AllHepMCInputManager("HEPMCIN");
    in->set_vertex_distribution_function(uniform,uniform,uniform);
    in->set_vertex_distribution_mean(0.0,0.0,0.0);
    in->set_vertex_distribution_width(0.0,0.0,5.0);
    se->registerInputManager( in );
    se->fileopen( in->Name().c_str(), inputFile );

    Fun4AllHepMCInputManager::VTXFUNC gaus = Fun4AllHepMCInputManager::Gaus;
    Fun4AllHepMCPileupInputManager *pileup = new Fun4AllHepMCPileupInputManager("PILEUPIN");
    pileup->set_vertex_distribution_function(gaus,gaus,gaus);
    pileup->set_vertex_distribution_mean(0.0,0.0,0.0);
    pileup->set_vertex_distribution_width(0.0,0.0,20.0):
    pileup->set_time_window(-18000.0,+18000.0); // ns
    pileup->set_collision_rate(100); // kHz
    se->registerInputManager( pileup );
    se->fileopen( pileup->Name().c_str(), pileupFile );
  }
```

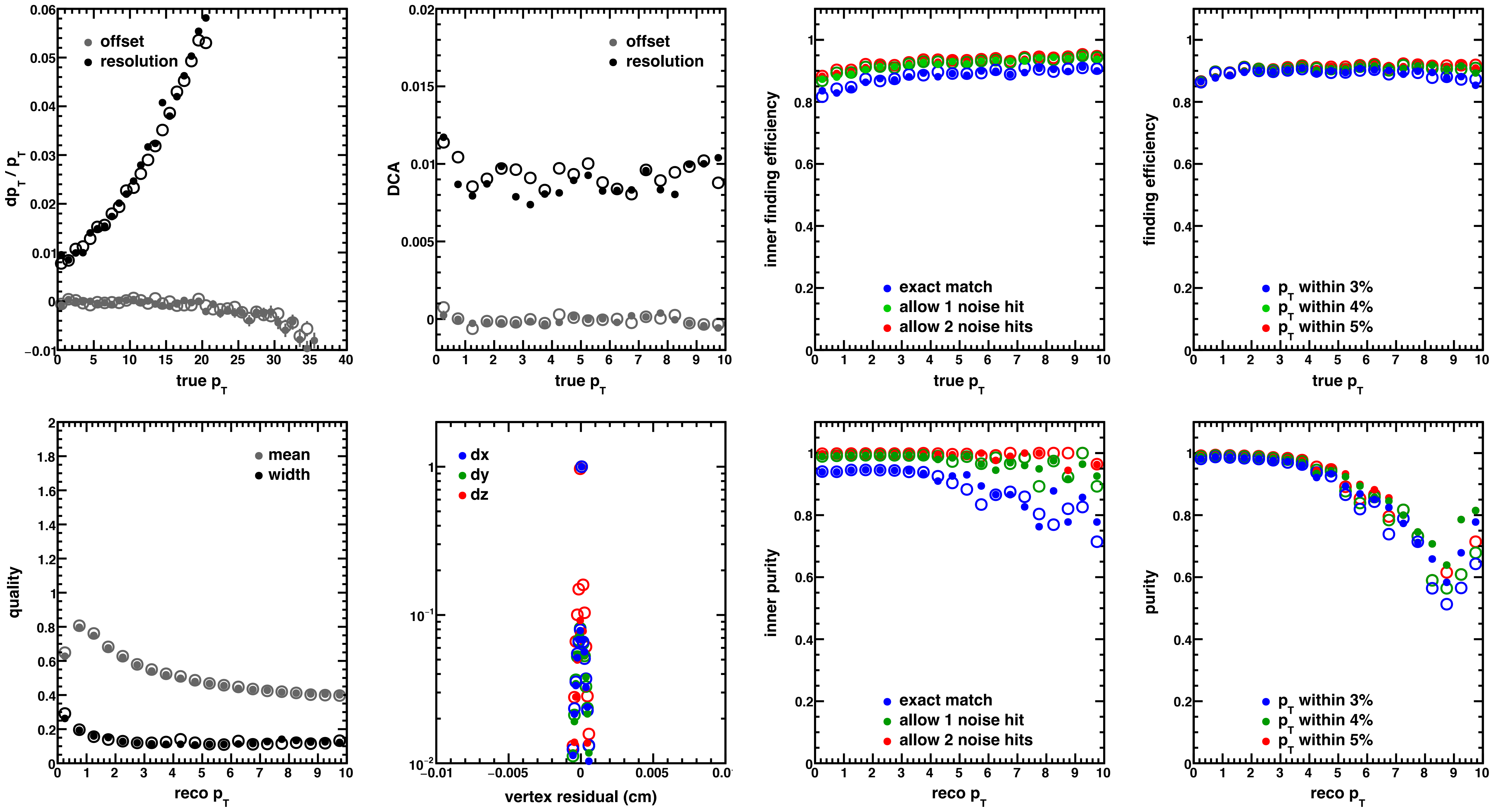
Example Macro: /phenix/u/mccumber/svtx/stage1_jobs/Fun4All_SvtxCheck.C

Input Files: /phenix/u/mccumber/svtx/stage1_jobs/in/{hijing_*.txt*,pileup_*.txt*}

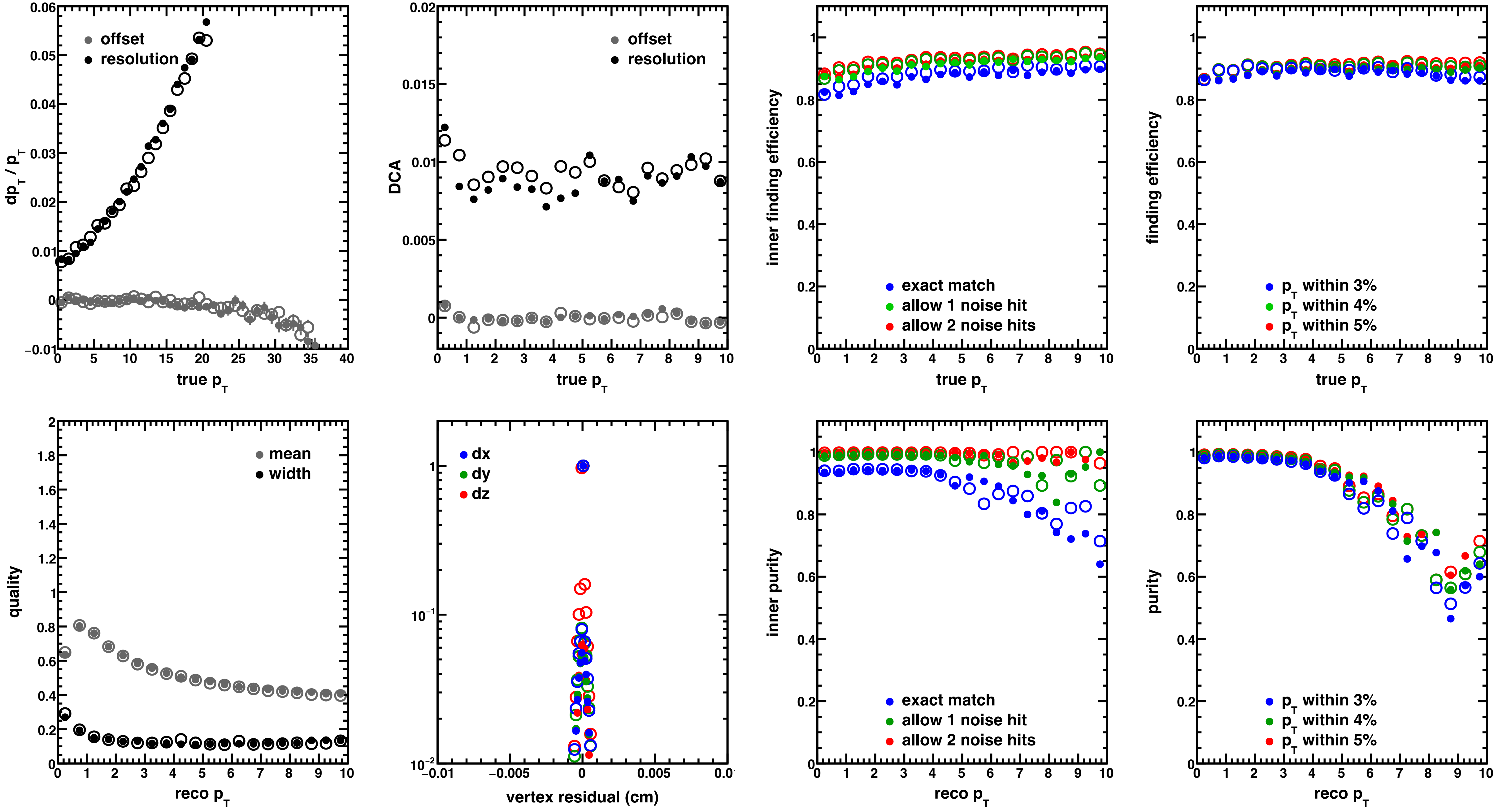
Au+Au Rate = 1 kHz



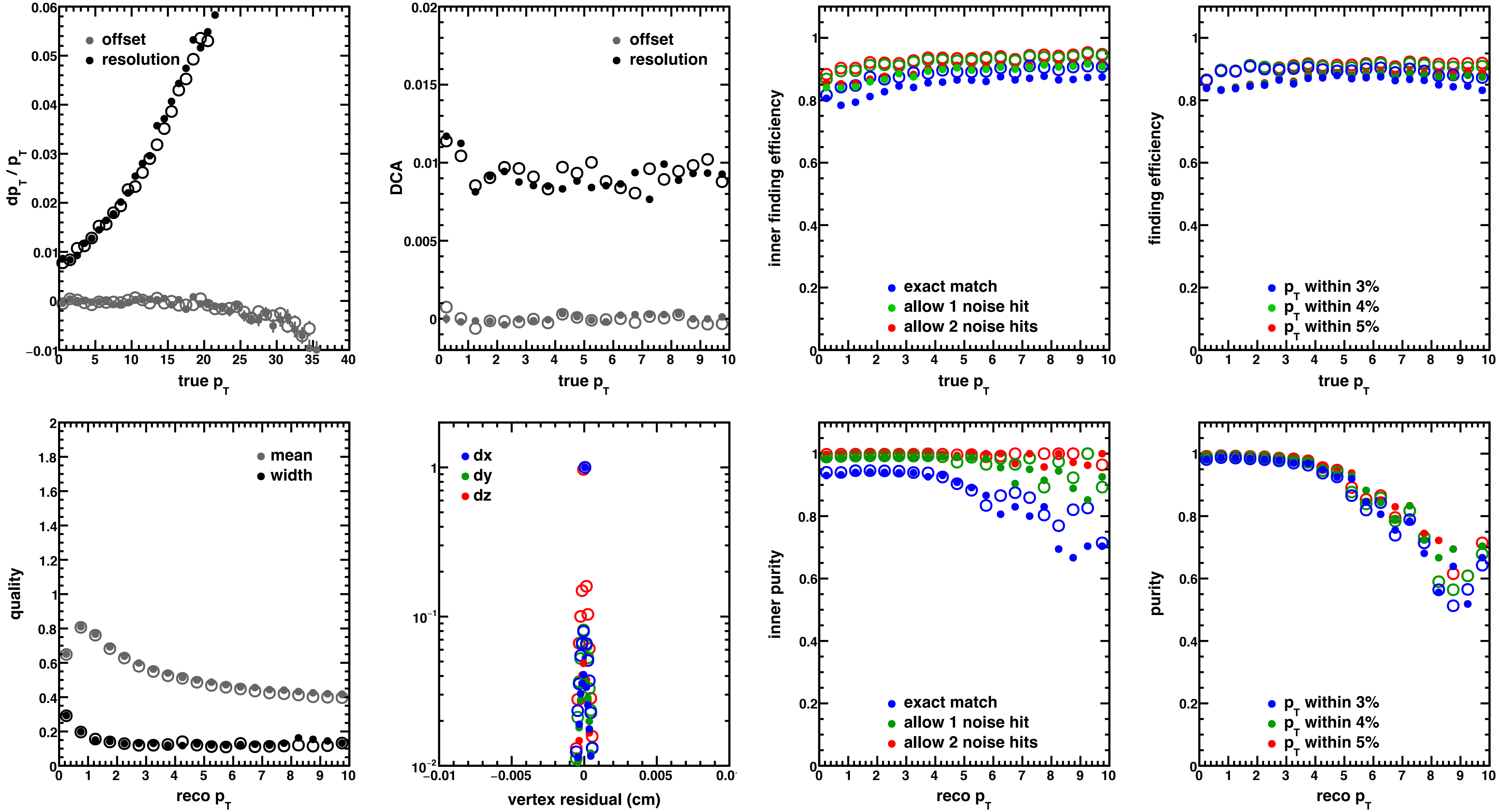
Au+Au Rate = 15 kHz



Au+Au Rate = 100 kHz



Au+Au Rate = 200 kHz



Primary effect is some modest track finding efficiency loss

Au+Au Rate = 300 kHz

I’m hitting a performance wall by 300 kHz. 75% of jobs fail. At 100 kHz only ~1% fail. I thought I might be running through to the end of the pileup file. So reused the pileup file by up to a factor 10, but the problem is must be resource usage.

Since this is above our instantaneous luminosity target any how, I plan to stop here…

Summary

Our first look into pileup considerations is positive. A 3-layer confirmation of the **generic tracking is appears robust in most of the expect luminosity range**.

However since the tracking tune performance is already degraded in central events compared to single particle reconstructions, **these studies will need to be repeated as the tracking software is improved** to see that the degradation is not hiding some important effects.

Also after the tracking is improved, we will **still need to explore the effects on heavy flavor identification**.

I need to solve the vertex embedding issue, and fully eliminate the old node storage: then I can issue a pull request bringing these features into the build.

BACKUP SLIDES